

■ Highest & Lowest Frequency Element – DSA Notes

■ Problem Statement

Given an array, find:

- The element with the **highest frequency**.
- The element with the **lowest frequency**.

If two elements have the same frequency, choose the **smallest number**.

■ Examples

Main Example:

Input: [4, 4, 1, 2, 2, 2, 3, 3, 3, 3]

Output: Highest = 3, Lowest = 1

Example 1:

Input: [1, 2, 2, 3, 3, 3]

Output: {3, 1}

Example 2:

Input: [4, 5, 5, 5, 6, 6]

Output: {5, 4}

Example 3:

Input: [10, 10, 10, 20, 20]

Output: {10, 20}

■ Core Logic

1. Sort the array so identical elements come together.
2. Traverse once to find:
 - Element with maximum frequency (mode)
 - Element with minimum frequency
3. On frequency ties → smallest number wins.

■ Dry Run (arr = [1, 2, 2, 3, 3, 3])

Element	Frequency	Observation
1	1	Least so far → 1
2	2	
3	3	Highest → 3

■ C++ Code

```

class Solution {
public:
    int findMode(vector &v;) {
        sort(v.begin(), v.end());
        int maxfreq = 1, mode = v[0], cf = 1;
        for (int i = 1; i < v.size(); i++) {
            if (v[i] == v[i-1]) cf++;
            else cf = 1;
            if (cf > maxfreq) {
                maxfreq = cf;
                mode = v[i];
            }
        }
        return mode;
    }

    int findLef(vector &v;) {
        int minFre = INT_MAX, least = v[0], cf = 1;
        for (int i = 1; i < v.size(); i++) {
            if (v[i] == v[i-1]) cf++;
            else {
                if (cf < minFre) { minFre = cf; least = v[i-1]; }
                cf = 1;
            }
        }
        if (cf < minFre) least = v.back();
        return least;
    }

    pair highestAndLowestFrequency(vector &arr;) {
        int hf = findMode(arr);
        int lf = findLef(arr);
        return {hf, lf};
    }
};

```

■ Time & Space Complexity

Time Complexity: $O(n \log n)$ — sorting

Space Complexity: $O(1)$ — no extra space

■ Key Notes

- Sorting simplifies frequency counting.
- Perfect for problems involving modes.
- Tie-breaking uses smallest element.